

QInfer: Statistical inference software for quantum applications

Christopher Granade^{1 2}, Christopher Ferrie^{1 2}, Ian Hincks^{3 4}, Steven Casagrande, Thomas Alexander^{4 5}, Jonathan Gross⁶, Michal Kononenko^{4 5}, and Yuval Sanders^{4 5 7}

¹ School of Physics, University of Sydney, Sydney, NSW, Australia

² Centre for Engineered Quantum Systems, University of Sydney, Sydney, NSW, Australia

³ Department of Applied Mathematics, University of Waterloo, Waterloo, ON, Canada

⁴ Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada

⁵ Department of Physics and Astronomy, University of Waterloo, Waterloo, ON, Canada

⁶ Center for Quantum Information and Control, University of New Mexico, Albuquerque, NM 87131-0001, USA

⁷ Department of Physics and Astronomy, Macquarie University, Sydney, NSW, Australia

October 4, 2016

Characterizing quantum systems through experimental data is critical to applications as diverse as metrology and quantum computing. Analyzing this experimental data in a robust and reproducible manner is made challenging, however, by the lack of readily-available software for performing principled statistical analysis. We improve the robustness and reproducibility of characterization by introducing an open-source library, QInfer, to address this need. Our library makes it easy to analyze data from tomography, randomized benchmarking, and Hamiltonian learning experiments either in post-processing, or online as data is acquired. QInfer also provides functionality for predicting the performance of proposed experimental protocols from simulated runs. By delivering easy-to-use characterization tools based on principled statistical analysis, QInfer helps address many outstanding challenges facing quantum technology.

Contents

1	Introduction	1
2	Inference and Particle Filtering	2
3	Applications in Quantum Information	4
3.1	Phase and Frequency Learning	4
3.2	State and Process Tomography	4
3.3	Randomized Benchmarking	5

4	Additional Functionality	6
4.1	Derived Models	6
4.2	Time-Dependent Models	8
4.3	Performance and Robustness Testing . .	9
4.4	Parallelization	10
4.5	Other Features	12
5	Conclusions	13
	Acknowledgments	13
	References	13
A	Custom Model Example	15

1 Introduction

Statistical modeling and parameter estimation play a critical role in many quantum applications. In quantum information in particular, the pursuit of large-scale quantum information processing devices has motivated a range of different characterization protocols, and in turn, new statistical models. For example, quantum state and process tomography are widely used to characterize quantum systems, and are in essence matrix-valued parameter estimation problems [1, 2]. Similarly, randomized benchmarking is now a mainstay in assessing quantum devices, motivating the use of rigorous statistical analysis [3] and algorithms [4]. Quantum metrology, meanwhile, is intimately concerned with what parameters can be extracted from measurements of a physical system, immediately necessitating a statistical view [5, 6].

Christopher Granade: cgranade@cgranade.com, www.cgranade.com

Complete source code available at DOI [10.5281/zenodo.157007](https://doi.org/10.5281/zenodo.157007).

The prevalence of statistical modeling in quantum applications should not be surprising: quantum mechanics is an inherently statistical theory, thus inference is an integral part of both experimental and theoretical practice. In the former, experimentalists need to model their systems and infer the value of parameters for the purpose of improving control as well as validating performance. In the latter, numerical experiments utilizing simulated data are now commonplace in theoretical studies, such that the same inference problems are encountered usually as a necessary step to answer questions about optimal data processing protocols or experiment design. In both cases, we lack tools to rapidly prototype and access inference strategies; **QInfer** addresses this need by providing a modular interface to a Monte Carlo algorithm for performing statistical inference.

Critically, in doing so, QInfer also supports and enables open and reproducible research practices. Parallel to the challenges faced in many other disciplines [7], physics research cannot long survive its own current practices. Open access, open source, and open data provide an indispensable means for research to be reproducible, ensuring that research work is useful to the communities invested in that research [8]. In the particular context of quantum information research, open methods are especially critical given the impact of statistical errors that can undermine the claims of published research [9, 10]. Ensuring the reproducibility of research is critical for evaluating the extent to which statistical and methodological errors undermine the credibility of published research [11].

QInfer also constitutes an important step towards a more general framework for quantum verification and validation (QCVV). As quantum information processor prototypes become more complex, the challenge of ensuring that noise processes affecting these devices conform to some agreed-upon standard becomes more difficult. This challenge can be managed, at least in principle, by developing confidence in the truth of certain simplifying assumptions and approximations. The value of randomized benchmarking, for example, depends strongly upon the extent to which noise is approximately Pauli [12]. QInfer provides a valuable framework for the design of automated and efficient noise assessment methods that will enable the comparison of actual device performance to the specifications demanded by theory.

To the end of enabling reproducible and accessible research, and hence providing a reliable process for interpreting advances in quantum information processing, we base QInfer using openly-available tools such as the Python programming language, the IPython interpreter, and Jupyter [13, 14]. Jupyter in particular

has already proven to be an invaluable tool for reproducible research, in that it provides a powerful framework for describing and explaining research software [15]. We provide our library under an open-source license along with examples [16] of how to use QInfer to support reproducible research practices. In this way, our library builds on and supports recent efforts to develop reproducible methods for physics research [17].

QInfer is a mature open-source software library written in the Python programming language which has now been extensively tested in a wide range of inferential problems by various research groups. Recognizing its maturity through its continuing development, we now formally release version 1.0. This maturity has given its developers the opportunity to step back and focus on the accessibility of QInfer such that other researchers can benefit from its utility. This short paper is the culmination of that effort. A full User’s Guide is available in the ancillary files.

We proceed as following. In Section 2, we give a brief introduction to Bayesian inference and particle filtering, the numerical algorithm we use to implement Bayesian updates. In Section 3, we describe applications of QInfer to common tasks in quantum information processing. Next, we describe in Section 4 additional features of QInfer before concluding in Section 5.

2 Inference and Particle Filtering

As QInfer is largely concerned with Bayesian approaches to statistical inference, we will briefly review the Bayesian formalism here before proceeding.

Statistical estimation begins with a model, which is a class of distributions of data D parameterized by model parameters x , written for example as $\{p_x(D)\}$. Thinking of this distribution operationally, we write it as $\Pr(D|x)$ and read it as “the probability of D given x .” The function $\Pr(D|x)$ is called the likelihood function, and computing it is equivalent to *simulating* an experiment. For example, the Born rule is a likelihood function, in that it maps a known quantum density matrix $x \equiv \rho$ to a distribution over measurement outcomes of a measurement $D \in \{E, \mathbb{1} - E\}$ via

$$\Pr(D = E|x) = \text{Tr}(E\rho). \quad (1)$$

The problem of estimating model parameters is an inverse problem. We are given a data set D and wish to estimate ρ . Bayes’ rule gives us what we need:

$$\Pr(x|D) = \frac{\Pr(D|x) \Pr(x)}{\Pr(D)}, \quad (2)$$

where $\Pr(x)$ is called the *prior* and $\Pr(x|D)$ is called the *posterior*. These distributions encode what we know

before and after doing an experiment. The denominator can be considered as a normalization constant that is usually computed implicitly in a numerical algorithm.

Importantly, we will demand that our data processing approach works in an *iterative* manner. Consider the example in which the data D is in fact a set $D = \{d_1, \dots, d_N\}$ of individual observations. In most if not all classical applications, each individual datum is distributed independently of the rest of the data set, conditioned on the true state. Formally, we write that for all j and k such that $j \neq k$, $d_j \perp d_k \mid x$. This may not hold in quantum models where measurement back-action can alter the state. In such cases, we can simply redefine what the parameters x label, such that this independence property can be taken as a convention, instead of as an assumption. Then, we have that

$$\Pr(x|d_1, \dots, d_N) = \frac{\Pr(d_N|x) \Pr(x|d_1, \dots, d_{N-1})}{\Pr(d_N)}. \quad (3)$$

In other words, the data can be processed *sequentially* where the prior for each successive datum is the posterior from the last.

This Bayes update can be solved analytically in some important special cases, such as frequency estimation [18, 19], but is more generally intractable. Thus, to develop a robust and generically useful framework for parameter estimation, we rely on numerical algorithms. In particular, QInfer is largely concerned with the *particle filtering* algorithm [20], also known as the sequential Monte Carlo (SMC) algorithm. In the context of quantum information, SMC was first proposed for learning from continuous measurement records [21], and has since been used to learn from state tomography [22], Hamiltonian learning [23], and randomized benchmarking [4], as well as other applications.

The idea of particle filtering is as follows. We represent the prior distribution as a weighted sum of delta functions,

$$\Pr(x) \approx \sum_k w_k \delta(x - x_k), \quad (4)$$

where the set $\{w_k\}$ are called the *weights* of the corresponding *particles* $\{x_k\}$. The posterior will also be a weighted sum of delta functions supported on the same particles. The weights, implied by Bayes' rule, are

$$w_k(D) = \frac{w_k \Pr(D|x_k)}{\sum_j w_j \Pr(D|x_j)}. \quad (5)$$

In practice, updating the weights in this fashion causes the particle filter to become unstable as data is collected; by default, QInfer will periodically apply the Liu-West algorithm to restore stability [24].

At any point during the processing of data, the expectation of any function with respect to the posterior is approximated as

$$\mathbb{E}[f(x)|D] \approx \sum_k w_k(D) f(x_k). \quad (6)$$

In particular, the expected error in x is given by the posterior covariance, $\text{Cov}(x|D) := \mathbb{E}[xx^T|D] - \mathbb{E}[x|D]\mathbb{E}^T[x|D]$. This can be used, for instance, to adaptively choose experiments which minimize the posterior variance [23]. This approach has been used to exponentially improve the number of samples required in frequency estimation problems [18, 19], and in phase estimation [25, 26]. Alternatively, other cost functions can be considered, such as the information gain [22, 27]. QInfer allows for quickly computing either the expected posterior variance or the information gain for proposed experiments, making it straightforward to develop adaptive experiment design protocols.

The functionality exposed by QInfer follows a simple object model, in which the experiment is described in terms of a *model*, and background information is described in terms of a *prior distribution*. These objects are then used with SMC to *update* the prior based on data. In particular, the iterative approach described above is formalized in terms of three Python classes:

qinfer.Distribution Represents drawing samples from a prior distribution.

qinfer.Model Represents evaluation of the likelihood function $\Pr(d|x; e)$ for a single datum d , a vector of model parameters x and an experiment e .

qinfer.SMCUpdater Implements updating a prior distribution to a posterior conditioned on a new datum, using particle filtering.

With the exception of `qinfer.SMCUpdater`, each of these classes is *abstract*, meaning that they specify the structure of the code rather than specific functionality. This allows the user to specify their own prior and likelihood function by either implementing these classes (as in the example of [Appendix A](#)), or by using one of the many concrete implementations provided with QInfer.

The concrete implementations provided with QInfer are useful in a range of common applications, as described in the next Section. We will demonstrate how these classes are used in practice with examples drawn from quantum information applications. We will also consider the `qinfer.Heuristic` class, which is useful in contexts such as online adaptive experiments and simulated experiments.

3 Applications in Quantum Information

In this Section, we describe various possible applications of QInfer to existing experimental protocols. In doing so, we highlight both functionality built-in to QInfer and how this functionality can be readily extended with custom models and distributions. We begin with the problems of phase and frequency learning, then describe the use of QInfer for state and process tomography, and conclude with applications to randomized benchmarking.

3.1 Phase and Frequency Learning

One of the primary applications for particle filtering is for learning the Hamiltonian H under which a quantum system evolves [23]. For instance, consider the single-qubit Hamiltonian $H = \omega\sigma_z/2$ for an unknown parameter ω . An experiment on this qubit may then consist of preparing a state $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$,

evolving for a time t and then measuring in the σ_x basis. This model commonly arises from Ramsey interferometry, and gives a likelihood function

$$\Pr(0|\omega; t) = |\langle + | e^{-i\omega t \sigma_z/2} | + \rangle|^2 = \cos^2(\omega t/2). \quad (7)$$

Note that this is also the same model for Rabi interferometry as well, with the interpretation of H as drive term rather than the internal Hamiltonian for a system. Similarly, this model forms the basis of Bayesian and maximum likelihood approaches to phase estimation.

In any case, QInfer implements (7) as the `SimplePrecessionModel` class, making it easy to quickly perform Bayesian inference for Ramsey or Rabi estimation problems. We demonstrate this in Listing 1, using `ExpSparseHeuristic` to select the k th measurement time $t_k = (9/8)^k$, as suggested by analytic arguments [18].

Listing 1: Frequency estimation example using `SimplePrecessionModel`.

```
>>> from qinfer import *
>>> model = SimplePrecessionModel()
>>> prior = UniformDistribution([0, 1])
>>> n_particles = 2000
5 >>> n_experiments = 100
>>> updater = SMCUpdater(model, n_particles, prior)
>>> heuristic = ExpSparseHeuristic(updater)
>>> true_params = prior.sample()
>>> for idx_experiment in range(n_experiments):
10 ...     experiment = heuristic()
...     datum = model.simulate_experiment(true_params, experiment)
...     updater.update(datum, experiment)
>>> print(updater.est_mean())
```

More complicated models for learning Hamiltonians with particle filtering have also been considered [28–31]; these can be readily implemented in QInfer as custom models by deriving from the `Model` class, as described in Appendix A.

3.2 State and Process Tomography

Though originally conceived of as an algebraic inverse problem, quantum tomography is also a problem of parameter estimation. Many have also considered the problem in a Bayesian framework [32, 33] and the sequential Monte Carlo algorithm has been used in both theoretical and experimental studies [22, 27, 34–36].

To define the model, we start with a basis for traceless Hermitian operators $\{B_j\}_{j=1}^{d^2-1}$. In the case of a qubit, this could be the basis of Pauli matrices, for example. Then, any state ρ can be written

$$\rho = \frac{\mathbb{1}}{d} + \sum_{j=1}^{d^2-1} \theta_j B_j, \quad (8)$$

for some vector of parameters θ . These parameters must be constrained such that $\rho \geq 0$.

In the simplest case, we can consider two-outcome measurements represented by the pair $\{E, \mathbb{1} - E\}$. The Born rule defines the likelihood function

$$\Pr(E|\rho) = \text{Tr}(\rho E). \quad (9)$$

For multiple measurements, we simply iterate. For many trials of the same measurement, we can use a *derived model* as discussed below.

QInfer’s `TomographyModel` abstracts many of the implementation details of this problem, exposing tomographic models and estimates in terms of QuTiP’s `Qobj` class [37]. This allows for readily integrating QInfer functionality with that of QuTiP, such as fidelity metrics, diamond norm calculation, and other such manipulations.

Tomography support in QInfer requires one of the bases mentioned above in order to parameterize the state. Many common choices of basis are included as `TomographyBasis` objects, such as the Pauli or Gell-Mann

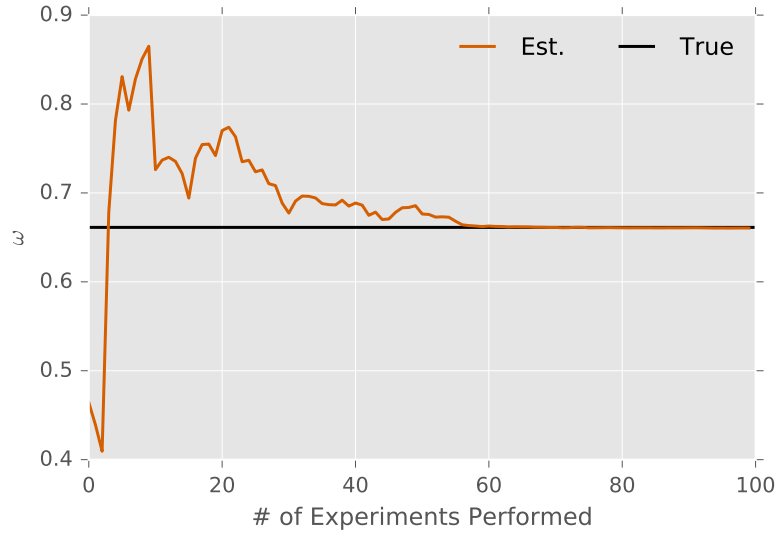


Figure 1: Frequency estimate obtained using Listing 1 as a function of the number of experiments performed.

bases. Many of the most commonly used priors are already implemented as a QInfer Distribution.

Listing 2: Rebit state tomography example using `TomographyModel`.

```
>>> from qinfer import *
>>> from qinfer.tomography import *
>>> basis = pauli_basis(1) # Single-qubit Pauli basis.
>>> model = TomographyModel(basis)
5 >>> prior = GinibreReditDistribution(basis)
>>> updater = SMCUpdater(model, 8000, prior)
>>> heuristic = RandomPauliHeuristic(updater)
>>> true_state = prior.sample()
>>>
10 >>> for idx_experiment in range(500):
>>>     experiment = heuristic()
>>>     datum = model.simulate_experiment(true_state, experiment)
>>>     updater.update(datum, experiment)
```

For simulations, common randomized measurement choices are already implemented. For example, `RandomPauliHeuristic` chooses random Pauli measurements for qubit tomography.

In Listing 2, we demonstrate QInfer’s tomography support for a *rebit*. By analogy to the Bloch sphere, a rebit may be represented by a point in the unit disk, making rebit tomography useful for plotting examples.

3.3 Randomized Benchmarking

In recent years, randomized benchmarking (RB) has reached a critical role in evaluating candidate quantum information processing systems. By using random sequences of gates drawn from the Clifford group, RB provides a likelihood function that depends on the fidelity with which each Clifford group element is implemented, allowing for estimates of that fidelity to be

drawn from experimental data [4].

In particular, suppose that each gate is implemented with fidelity F , and consider a fixed initial state and measurement. Then, the survival probability over sequences of length m is given by [38]

$$\Pr(\text{survival}|p, A, B; m) = Ap^m + B, \quad (10)$$

where $p := (dF - 1)/(d - 1)$, d is the dimension of the system under consideration, and where A and B describe the state preparation and measurement (SPAM) errors. Learning the model $x = (p, A, B)$ thus provides an estimate of the fidelity of interest F .

The likelihood function for randomized benchmarking is extremely simple, and requires only scalar arithmetic to compute, making it especially useful for avoiding the computational overhead typically required to characterize large quantum systems with classical resources. Multiple generalizations of RB

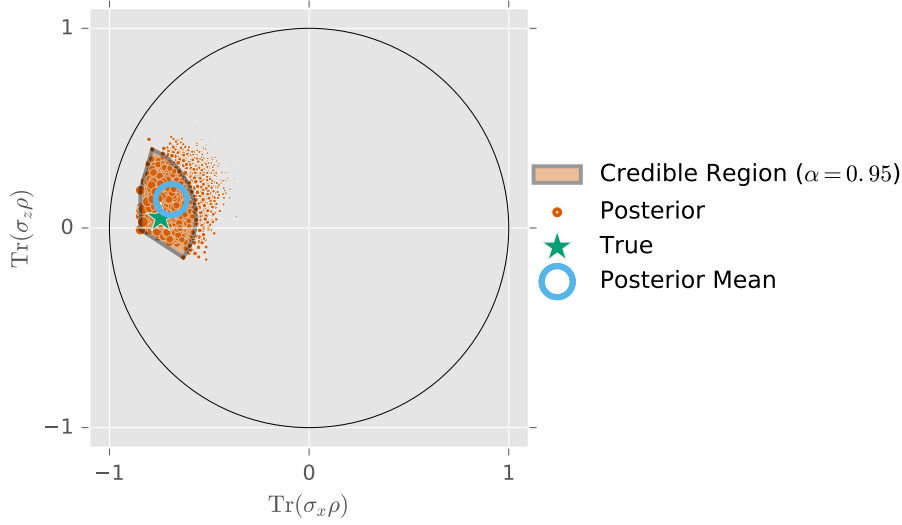


Figure 2: Posterior over rebit states after 100 random Pauli measurements, each repeated five times, as implemented by [Listing 2](#).

have been recently developed which extend these benefits to estimating crosstalk [39], coherence [40], and to estimating fidelities of non-Clifford gates [41, 42]. RB has also been extended to provide tomographic information as well [43]. The estimates provided by randomized benchmarking have also been applied to design improved control sequences [44, 45].

QInfer supports RB experiments through the `qinfer.RandomizedBenchmarkingModel` class. For common priors, QInfer also provides a simplified interface, `qinfer.simple_est_rb`, that reports the mean and covariance over an RB model given experimental data. We provide an example in [Listing 3](#).

Listing 3: Randomized benchmarking example using `simple_est_rb`.

```
>>> from qinfer import *
>>> import numpy as np
>>> p, A, B = 0.95, 0.5, 0.5
>>> ms = np.linspace(1, 800, 201).astype(int)
5 >>> signal = A * p ** ms + B
>>> n_shots = 25
>>> counts = np.random.binomial(p=signal, n=n_shots)
>>> data = np.column_stack([counts, ms, n_shots * np.ones_like(counts)])
>>> mean, cov = simple_est_rb(data, n_particles=12000, p_min=0.8)
10 >>> print(mean, np.sqrt(np.diag(cov)))
```

4 Additional Functionality

Having introduced common applications for QInfer, in this Section we describe additional functionality which can be used with each of these applications, or with custom models.

4.1 Derived Models

QInfer allows for the notion of a *model chain*, where the likelihood of a given model in the chain is a function of the likelihoods of models below it, and possibly new model or experiment parameters. This abstraction is useful for a couple of reasons. It encourages more ro-

bust programs, since models in the chain will often be debugged independently. It also often makes writing new models easier since part of the chain may be included by default in the QInfer library, or may overlap with other similar models the user is implementing. Finally, in quantum systems, it is common to have a likelihood function which is most naturally expressed as a hierarchical probability distribution, with base models describing quantum physics, and overlying models describing measurement processes.

Model chains are typically implemented through the use of the abstract class `DerivedModel`. Since this class itself inherits from the `Model` class, subclass instances must provide standard model properties and meth-

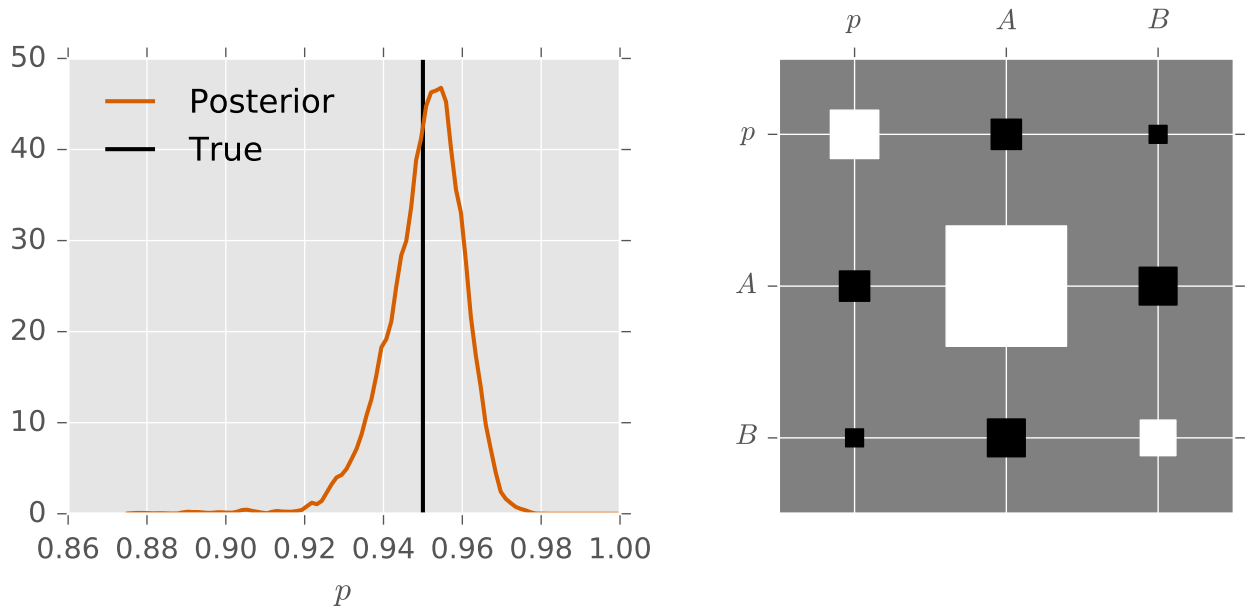


Figure 3: (Left) Posterior over the randomized benchmarking decay rate parameter p after measuring 25 sequences at each of 201 sequence lengths, as described in Listing 3. (Right) The posterior covariance matrix over all three randomized benchmarking parameters $\mathbf{x} = (p, A, B)$, represented as a Hinton diagram. White squares indicate positive elements, while black squares indicate negative elements, and the relative sizes indicate magnitude of each element.

ods such as `likelihood`, `n_outcomes`, and `modelparam_names`. Additionally, `DerivedModel` accepts an argument `model`, referring to the *underlying model* directly below it in the model chain. Class properties exist for referencing models at arbitrary depths in the chain, all the way down to the *base model*.

As an example, consider a base model which is the precession model discussed in Section 3.1. This is a two-outcome model whose outcomes correspond to measuring the state $|+\rangle$ or the orthogonal state $|-\rangle$,

which can be viewed as flipping a biased coin. Perhaps an actual experiment of this system consists of flipping the coin N times with identical settings, where the individual results are not recorded, only the total number n_+ of $|+\rangle$ results. In this case, we can concatenate this base model with the built-in `DerivedModel` called `BinomialModel`. This model adds an additional experiment parameter `n_meas` specifying how many times the underlying model’s coin is flipped in a single experiment.

Listing 4: Frequency estimation with the derived `BinomialModel` and a linear time-sampling heuristic.

```
>>> from qinfer import *
>>> import numpy as np
>>> model = BinomialModel(SimplePrecessionModel())
>>> n_meas = 25
5 >>> prior = UniformDistribution([0, 1])
>>> updater = SMCUpdater(model, 2000, prior)
>>> true_params = prior.sample()
>>> for t in np.linspace(0.1, 20, 20):
...     experiment = np.array([(t, n_meas)], dtype=model.expparams_dtype)
10 ...     datum = model.simulate_experiment(true_params, experiment)
...     updater.update(datum, experiment)
>>> print(updater.est.mean())
```

Note that parallelization, discussed in Section 4.4, is implemented as a `DerivedModel` whose likelihood batches the underlying model’s likelihood function

across processors.

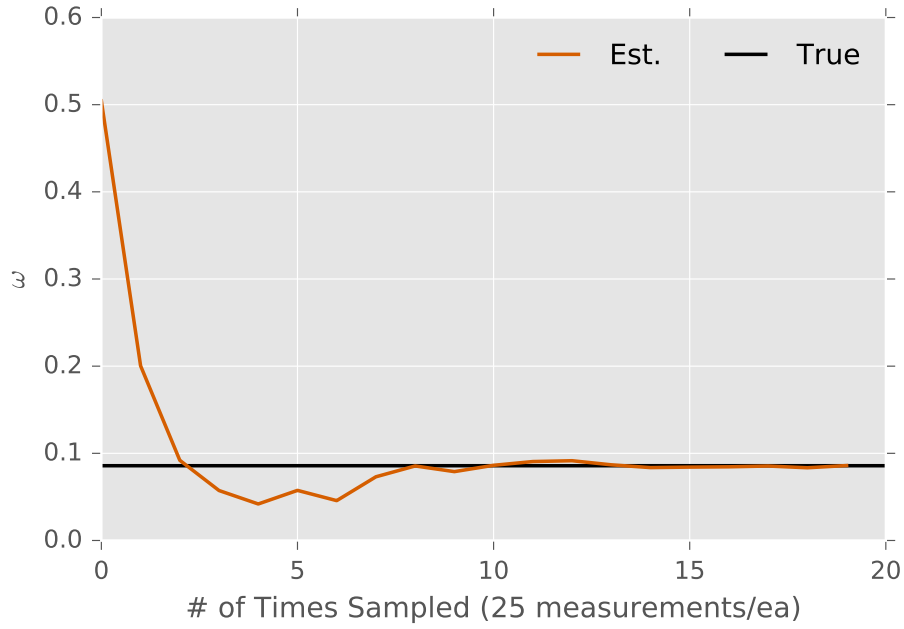


Figure 4: Frequency estimate after 25 measurements at each of 20 linearly-spaced times, using `qinfer.BinomialModel` as in Listing 4.

4.2 Time-Dependent Models

So far, we have only considered time-independent (parameter estimation) models, but particle filtering is useful more generally for estimating time-dependent (state-space) models. Following the work of Isard and Blake [46], when performing a Bayes update, we can also incorporate state-space dynamics by adding a time-step update. For example, to follow a Wiener process, we move each particle $x_i(t_k)$ at time t_k to its new position

$$x_i(t_{k+1}) = x_i(t_k) + (t_{k+1} - t_k)\eta, \quad (11)$$

with $\eta \sim N(0, \Sigma)$ for a covariance matrix Σ .

In QInfer, this is handled by implementing the `Model.update_timestep` method, which specifies the time step that `SMCUpdater` should perform after each datum. This design allows for the specification of more complicated time step updates than the example of (11). For instance, the `qinfer.RandomWalkModel` class adds diffusive steps to existing models and can be used to quickly learn time-dependent properties, such as shown in Listing 5. Moreover, QInfer provides the `DiffusiveTomographyModel` for including time-dependence in tomography by truncating time step updates to lie within the space of valid states [35]. A video example of time-dependent tomography can be found on YouTube [47].

Listing 5: Frequency estimation with a time-dependent model.

```

>>> from qinfer import *
>>> import numpy as np
>>> prior = UniformDistribution([0, 1])
>>> true_params = np.array([[0.5]])
5 >>> n_particles = 2000
>>> model = RandomWalkModel(
...     BinomialModel(SimplePrecessionModel()), NormalDistribution(0, 0.01**2))
>>> updater = SMCUpdater(model, n_particles, prior)
>>> t = np.pi / 2
10 >>> n_meas = 40
>>> expparams = np.array([(t, n_meas)], dtype=model.expparams.dtype)
>>> for idx in range(1000):
...     datum = model.simulate_experiment(true_params, expparams)
...     true_params = np.clip(model.update_timestep(true_params, expparams)[: , : , 0], 0, 1)
15 ...     updater.update(datum, expparams)

```

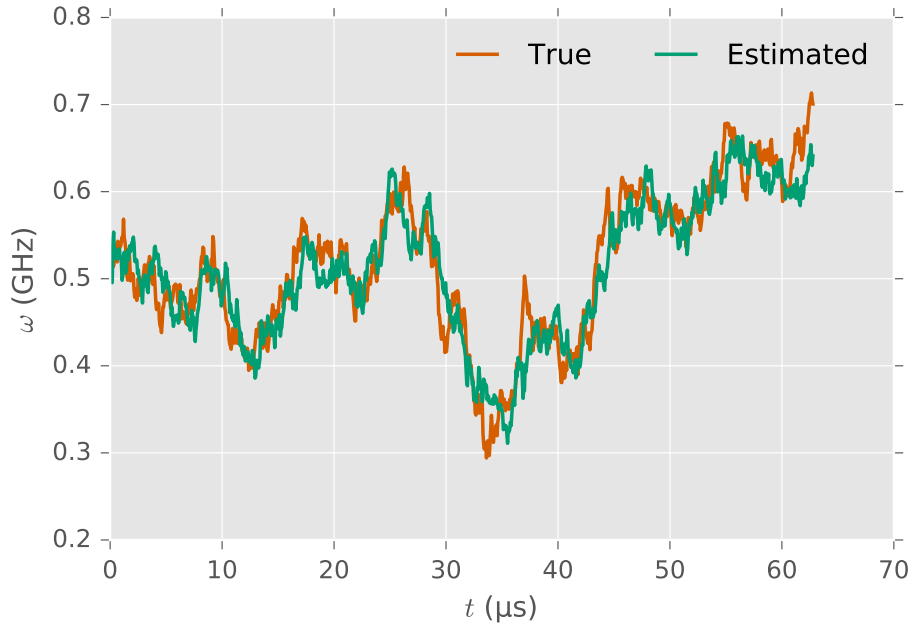


Figure 5: Time-dependent frequency estimation, using `qinfer.RandomWalkModel` as in [Listing 5](#).

4.3 Performance and Robustness Testing

One important application of QInfer is predicting how well a particular parameter estimation experiment will work in practice. This can be formalized by considering the risk $R(x) := \mathbb{E}_D[(\hat{x}(D) - x)^T(\hat{x}(D) - x)]$ incurred by the estimate $\hat{x}(D)$ as a function of some true model x . The risk can be estimated by drawing many different data sets D , computing the estimates for each, and reporting the average error. Similarly, one can estimate the Bayes risk $r(\pi) := \mathbb{E}_{x \sim \pi}[R(x)]$ by drawing a new “true” model x from a prior π along with each data set.

In both cases, QInfer automates the process of performing many independent estimation trials through the `perf.test.multiple` function. This function will run an updater loop for a given model, prior, and experiment design heuristic, returning the errors incurred after each measurement in each trial. Taking an expecta-

tion value with `numpy.mean` returns the risk or Bayes risk, depending if the `true.model` keyword argument is set.

For example, the following snippet finds the Bayes risk for a frequency estimation experiment ([Section 3.1](#)) as a function of the number of measurements performed.

Performance evaluation can also easily be parallelized over trials, as discussed in [Section 4.4](#), allowing for efficient use of computational resources. This is especially important when comparing performance for a range of different parameters. For instance, one might want to consider how the risk and Bayes risk of an estimation procedure scale with errors in a faulty simulator; QInfer supports this usecase with the `qinfer.PoisonedModel` derived model, which adds errors to an underlying “valid” model. In this way, QInfer enables quickly reasoning about how much approximation error can be tolerated by an estimation procedure.

Listing 6: Bayes risk of frequency estimation as a function of the number of measurements, calculated using `perf.test.multiple`.

```
>>> performance = perf.test.multiple(
...     # Use 100 trials to estimate expectation over data.
...     100,
...     # Use a simple precession model both to generate,
5 ...     # data, and to perform estimation.
...     SimplePrecessionModel(),
...     # Use 2,000 particles and a uniform prior.
...     2000, UniformDistribution([0, 1]),
```

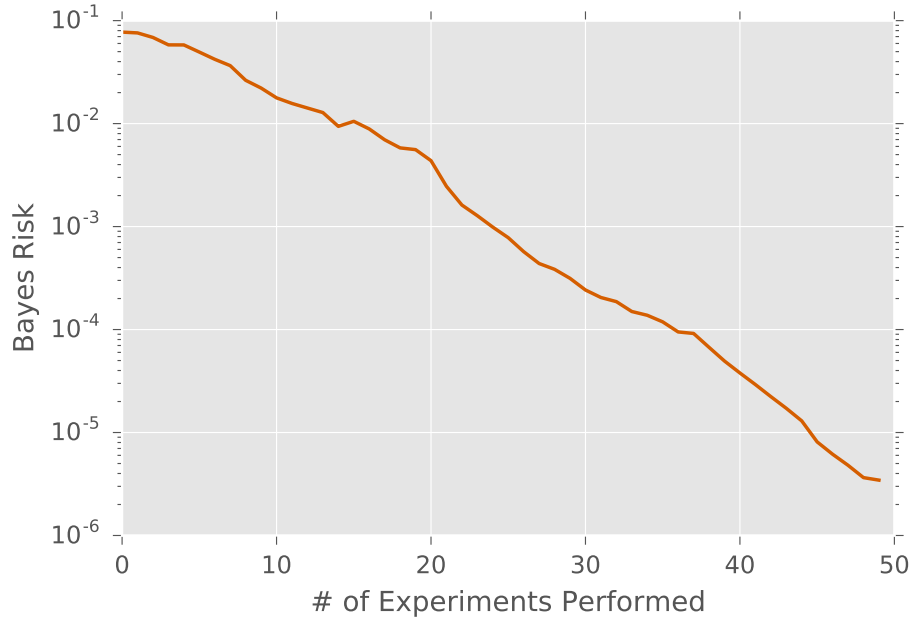


Figure 6: Bayes risk of a frequency estimation model with exponentially sparse sampling as a function of the number of experiments performed, and as calculated by Listing 6.

```

...     # Take 50 measurements with  $t_k = ab^k$ .
10 ...     50, ExpSparseHeuristic
... )
>>> # The returned performance data has an index for the trial, and an index for the measurement number.
>>> print(performance.shape)
(100, 50)
15 >>> # Calculate the Bayes risk by taking a mean over the trial index.
>>> risk = np.mean(performance['loss'], axis=0)

```

4.4 Parallelization

At each step of the SMC algorithm, the likelihood $\Pr(d_n|\mathbf{x})$ of an experimental datum d_n is computed for every particle \mathbf{x}_k in the distribution. Typically, the total running time of the algorithm is overwhelmingly spent calculating these likelihoods. However, individual likelihood computations are independent of each other and therefore may be performed in parallel. On a single computational node with multiple cores, limited parallelization is performed automatically by relying on NumPy’s vectorization primitives [48].

More generally, however, if the running time of $\Pr(d_n|\mathbf{x})$ is largely independent of \mathbf{x} , we may divide

our particles into L disjoint groups,

$$\{\mathbf{x}_1^{(1)}, \dots, \mathbf{x}_{k_1}^{(1)}\} \sqcup \dots \sqcup \{\mathbf{x}_1^{(L)}, \dots, \mathbf{x}_{k_L}^{(L)}\}, \quad (12)$$

and send each group along with d_n to a separate processor to be computed in parallel.

In QInfer, this is handled by the derived model (Section 4.1) `qinfer.DirectViewParallelizedModel` which uses the Python library `ipyparallel` [49]. This library supports everything from simple parallelization over the cores of a single processor, to make-shift clusters set up over SSH, to professional clusters using standard job schedulers. Passing the model of interest as well as an `ipyparallel.DirectView` of the processing engines is all that is necessary to parallelize a model.

Listing 7: Example of parallelizing likelihood calls with `DirectViewParallelizedModel`.

```

>>> from qinfer import *
>>> from ipyparallel import Client
>>> rc = Client(profile="my_cores")
>>> model = DirectViewParallelizedModel(SimplePrecessionModel(), rc[:])

```

In Figure 7, a roughly $12\times$ speed-up is demonstrated

by parallelizing a model over the 12 cores of a sin-

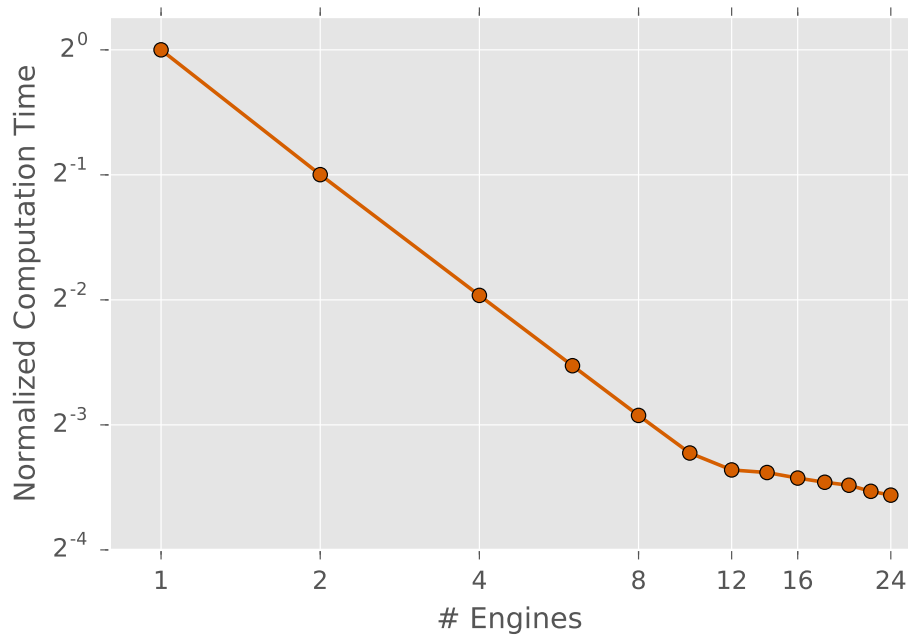


Figure 7: Parallelization of the likelihood function being tested on a single computer with 12 physical Intel Xeon cores. 5000 particles are shared over a varying number of ipyparallel engines. The linear unit slope indicates that overhead is negligible in this example. This holds until the number of physical cores is reached, past which hyper-threading continues to give diminishing returns. The single-engine running time was about 37 seconds, including ten different experiment values, and 5 possible outcomes.

gle computer. This model was contrived to demonstrate the parallelization potential of a generic Hamiltonian learning problem which uses dense operators and states. A single likelihood call generates a random 16×16 anti-hermitian matrix (representing the generator of a four qubit system), exponentiates it, and returns overlap with the $|0000\rangle$ state. Implementation details can be found in the QInfer examples repository [16], or in the ancillary files.

So far, we have discussed parallelization from the perspective of traditional processors (CPUs), which typically have a small number of processing cores on each chip. By contrast, moderately-priced desktop graphical processing units (GPUs) will often contain thousands of cores, while GPU hosts tailored for scientific use can have tens of thousands. This massive parallelization makes GPUs attractive for particle filtering [50]. Using libraries such as PyOpenCL and PyCUDA [51] or Numba [52], custom models can be written which take advantage of GPUs within QInfer [53]. For example, `qinfer.AcceleratedPrecessionModel` offloads its computation of \cos^2 to GPUs using PyOpenCL.

4.5 Other Features

In addition to the functionality described above, QInfer has a wide range of other features that we describe more briefly here. A complete description can be found in the provided Users' Guide (see ancillary files or docs.qinfer.org).

Plotting and Visualization QInfer provides plotting and visualization support based on matplotlib [54] and mpltools [55]. In particular, `qinfer.SMCUpdater` provides methods for plotting posterior distributions and covariance matrices. These methods make it straightforward to visually diagnose the operation of and results obtained from particle filtering.

Similarly, the `qinfer.tomography` module provides several functions for producing plots of states and distributions over rebits (qubits restricted to real numbers). Rebit visualization is in particular useful for demonstrating the conceptual operation of particle filter-based tomography in a clear and attractive manner.

Fisher Information Calculation In evaluating estimation protocols, it is important to establish a baseline of how accurately one can estimate a model even in principle. Similarly, such a baseline can be used to compare between protocols by informing as to how much information can be extracted from a proposed experiment. The Cramér-Rao bound and its Bayesian analog, the van Trees inequality, formalize this notion in terms of the Fisher information matrix [56, 57]. For

any model which specifies its derivative in terms of a *score*, QInfer will calculate each of these bounds, providing useful information about proposed experimental and estimation protocols. The `qinfer.ScoreMixin` class builds on this by calculating the score of an arbitrary model using numerical differentiation.

Region Estimation As an alternative to specifying the entire posterior distribution approximated by `qinfer.SMCUpdater`, we provide methods for reporting credible regions over the posterior, based on covariance ellipsoids, convex hulls, and minimum volume enclosing ellipsoids [34]. These region estimators provide a rigorous way of summarizing one's uncertainty following an experiment (colloquially referred to as "error bars"), and owing to the Bayesian approach, do so in a manner consistent with experimental experience.

Model Selection and Averaging Statistical inference does not require asserting *a priori* the correctness of a particular model (that is, likelihood function), but allows a model to be taken as a hypothesis and compared to other models. This is made formal by *model selection*. From a Bayesian perspective, the ratio of the posterior normalizations for two different models gives a natural and principled model selection criterion, known as the Bayes factor [58]. The Bayes factor provides a model selection rule that is significantly more robust to outlying data than conventional hypothesis testing approaches [59]. For example, in quantum applications, the Bayes factor is particularly useful in tomography, and can be used to decide the rank or dimension of a state [34]. QInfer implements this criterion as the `SMCUpdater.normalization.record` property, allowing for model selection and averaging to be performed in a straightforward manner.

Approximate Maximum-Likelihood Estimation As opposed to the Bayesian approach, one may also consider *maximum likelihood* estimation (MLE), in which a model is estimated as $\hat{x}_{\text{MLE}} := \arg \max_x \Pr(D|x)$. MLE can be approximated as the mean of an artificially tight posterior distribution obtained by performing Bayesian inference with a likelihood function $\Pr'(D|x)$ related to the true likelihood by

$$\Pr'(D|x) = (\Pr(D|x))^\gamma \quad (13)$$

for a quality parameter $\gamma > 1$ [60]. In QInfer, this is implemented by the class `qinfer.MLEModel`, which decorates another model in the manner of [Section 4.1](#).

Likelihood-Free Estimation For some models, explicitly calculating the likelihood function $\Pr(D|x)$ is intractable, but good approaches may exist for drawing new data sets consistent with a hypothesis. This is the case, for instance, if a simulator is implemented using quantum resources. In the absence of an explicit likelihood function, Bayesian inference must be implemented in a *likelihood-free* manner, using hypothetical data sets consistent to form an approximate likelihood instead [61]. This introduces an estimation error which can be modeled in QInfer by using the `qinfer.PoisonedModel` class discussed in Section 4.3.

Simplified Estimation For the frequency estimation and randomized benchmarking examples described in Section 3, QInfer provides functions to perform estimation using a “standard” updater loop, making it easy to load data from NumPy-, MATLAB- or CSV-formatted files.

Jupyter Integration Several QInfer classes, including `qinfer.Model` and `qinfer.SMCUpdater`, integrate with Jupyter Notebook to provide additional information formatted using HTML. Moreover, the `qinfer.IPythonProgressBar` class provides a progress bar as a Jupyter Notebook widget with a QtIP-compatible interface, making it easy to report on performance testing progress.

MATLAB/Julia Interoperability Finally, QInfer functionality is also compatible with MATLAB 2016a and later, and with Julia (using the `PyCall.jl` package [62]), enabling integration both with legacy code and with new developments in scientific computing.

5 Conclusions

In this work, we have presented QInfer, our open-source library for statistical inference in quantum information processing. QInfer is useful for a range of different applications, and can be readily used for custom problems due to its modular and extensible design, addressing a pressing need in both quantum information theory and in experimental practice. Importantly, our library is also accessible, in part due to the extensive documentation that we provide (see ancillary files or docs.qinfer.org). In this way, QInfer supports the goal of reproducible research by providing open-source tools for data analysis in a clear and understandable manner.

Acknowledgments

CG and CF acknowledge funding from the Army Research Office grant numbers W911NF-14-1-0098 and W911NF-14-1-0103, from the Australian Research Council Centre of Excellence for Engineered Quantum Systems. CG, CF, IH, and TA acknowledge funding from Canadian Excellence Research Chairs (CERC), Natural Sciences and Engineering Research Council of Canada (NSERC), the Province of Ontario, and Industry Canada. YS acknowledges funding from ARC Discovery Project DP160102426. CG greatly appreciates help in testing and feedback from Nathan Wiebe, Joshua Combes, and Alan Robertson.

References

- [1] G. M. D’Ariano, M. D. Laurentis, M. G. A. Paris, A. Porzio, and S. Solimeno, “Quantum tomography as a tool for the characterization of optical devices,” *Journal of Optics B: Quantum and Semi-classical Optics* **4**, S127 (2002).
- [2] J. B. Altepeter, D. Branning, E. Jeffrey, T. C. Wei, P. G. Kwiat, R. T. Thew, J. L. O’Brien, M. A. Nielsen, and A. G. White, “Ancilla-assisted quantum process tomography,” *Phys. Rev. Lett.* **90**, 193601 (2003).
- [3] J. J. Wallman and S. T. Flammia, “Randomized benchmarking with confidence,” *New Journal of Physics* **16**, 103032 (2014).
- [4] C. Granade, C. Ferrie, and D. G. Cory, “Accelerated randomized benchmarking,” *New Journal of Physics* **17**, 013042 (2015).
- [5] A. S. Holevo, *Statistical Structure of Quantum Theory*, edited by R. Beig, J. Ehlers, U. Frisch, K. Hepp, W. Hillebrandt, D. Imboden, R. L. Jaffe, R. Kippenhahn, R. Lipowsky, H. v. Löhneysen, I. Ojima, H. A. Weidenmüller, J. Wess, J. Zittartz, and W. Beiglböck, Lecture Notes in Physics Monographs, Vol. 67 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2001).
- [6] C. W. Helstrom, *Quantum Detection and Estimation Theory* (Academic Press, 1976).
- [7] B. Goldacre, “Scientists are hoarding data and it’s ruining medical research,” *BuzzFeed* (2015).
- [8] V. Stodden and S. Miguez, “Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research,” *Journal of Open Research Software* **2** (2014), 10.5334/jors.ay.
- [9] J. P. A. Ioannidis, “Why Most Published Research Findings Are False,” *PLOS Med* **2**, e124 (2005).
- [10] R. Hoekstra, R. D. Morey, J. N. Rouder, and E.-J.

- Wagenmakers, "Robust misinterpretation of confidence intervals," *Psychonomic Bulletin & Review*, 1 (2014).
- [11] J. P. A. Ioannidis, "How to Make More Published Research True," *PLOS Med* **11**, e1001747 (2014).
- [12] Y. R. Sanders, J. J. Wallman, and B. C. Sanders, "Bounding quantum gate error rate based on reported average fidelity," *New Journal of Physics* **18**, 012002 (2016).
- [13] F. Pérez and B. E. Granger, "IPython: A System for Interactive Scientific Computing," *Computing in Science and Engineering* **9**, 21 (2007).
- [14] Jupyter Development Team, "Jupyter," (2016).
- [15] S. R. Piccolo and M. B. Frampton, "Tools and techniques for computational reproducibility," *GigaScience* **5**, 30 (2016); A. de Vries, "Using R with Jupyter Notebooks," (2015); D. Donoho and V. Stodden, "Reproducible research in the mathematical sciences," in *The Princeton Companion to Applied Mathematics*, edited by N. J. Higham (2015).
- [16] C. Granade, C. Ferrie, I. Hincks, S. Casagrande, T. Alexander, J. Gross, M. Kononenko, and Y. Sanders, "QInfer Examples," <http://goo.gl/4sXY1t>.
- [17] M. Dolfi, J. Gukelberger, A. Hehn, J. Imriška, K. Pakrouski, T. F. Rønnow, M. Troyer, I. Zintchenko, F. Chirigati, J. Freire, and D. Shasha, "A model project for reproducible papers: Critical temperature for the Ising model on a square lattice," [arXiv:1401.2000 \[cond-mat, physics:physics\]](https://arxiv.org/abs/1401.2000) (2014), [arXiv:1401.2000 \[cond-mat, physics:physics\]](https://arxiv.org/abs/1401.2000).
- [18] C. Ferrie, C. E. Granade, and D. G. Cory, "How to best sample a periodic probability distribution, or on the accuracy of Hamiltonian finding strategies," *Quantum Information Processing* **12**, 611 (2013).
- [19] A. Sergeevich, A. Chandran, J. Combes, S. D. Bartlett, and H. M. Wiseman, "Characterization of a qubit Hamiltonian using adaptive measurements in a fixed basis," *Physical Review A* **84**, 052315 (2011).
- [20] A. Doucet and A. M. Johansen, *A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later* (2011).
- [21] B. A. Chase and J. M. Geremia, "Single-shot parameter estimation via continuous quantum measurement," *Physical Review A* **79**, 022314 (2009).
- [22] F. Huszár and N. M. T. Houlby, "Adaptive Bayesian quantum tomography," *Physical Review A* **85**, 052120 (2012).
- [23] C. E. Granade, C. Ferrie, N. Wiebe, and D. G. Cory, "Robust online Hamiltonian learning," *New Journal of Physics* **14**, 103013 (2012).
- [24] J. Liu and M. West, "Combined parameter and state estimation in simulation-based filtering," in *Sequential Monte Carlo Methods in Practice*, edited by De Freitas and N. Gordon (Springer-Verlag, New York, 2001).
- [25] D. W. Berry and H. M. Wiseman, "Optimal States and Almost Optimal Adaptive Measurements for Quantum Interferometry," *Physical Review Letters* **85**, 5098 (2000).
- [26] B. L. Higgins, D. W. Berry, S. D. Bartlett, H. M. Wiseman, and G. J. Pryde, "Entanglement-free Heisenberg-limited phase estimation," *Nature* **450**, 393 (2007).
- [27] G. I. Struchalin, I. A. Pogorelov, S. S. Straupe, K. S. Kravtsov, I. V. Radchenko, and S. P. Kulik, "Experimental adaptive quantum tomography of two-qubit states," *Physical Review A* **93**, 012103 (2016).
- [28] N. Wiebe, C. Granade, C. Ferrie, and D. G. Cory, "Hamiltonian learning and certification using quantum resources," *Physical Review Letters* **112**, 190501 (2014).
- [29] N. Wiebe, C. Granade, C. Ferrie, and D. Cory, "Quantum Hamiltonian learning using imperfect quantum resources," *Physical Review A* **89**, 042314 (2014).
- [30] N. Wiebe, C. Granade, and D. G. Cory, "Quantum bootstrapping via compressed quantum Hamiltonian learning," *New Journal of Physics* **17**, 022005 (2015).
- [31] M. P. V. Stenberg, Y. R. Sanders, and F. K. Wilhelm, "Efficient Estimation of Resonant Coupling between Quantum Systems," *Physical Review Letters* **113**, 210404 (2014).
- [32] K. R. W. Jones, "Principles of quantum inference," *Annals of Physics* **207**, 140 (1991).
- [33] R. Blume-Kohout, "Optimal, reliable estimation of quantum states," *New Journal of Physics* **12**, 043034 (2010).
- [34] C. Ferrie, "Quantum model averaging," *New Journal of Physics* **16**, 093035 (2014).
- [35] C. Granade, J. Combes, and D. G. Cory, "Practical Bayesian tomography," *New Journal of Physics* **18**, 033024 (2016).
- [36] K. S. Kravtsov, S. S. Straupe, I. V. Radchenko, N. M. T. Houlby, F. Huszár, and S. P. Kulik, "Experimental adaptive Bayesian tomography," *Physical Review A* **87**, 062122 (2013).
- [37] A. Pitchford, C. Granade, P. D. Nation, and R. J. Johansson, "QuTiP 3.2.0," (2015–).
- [38] E. Magesan, J. M. Gambetta, and J. Emerson, "Scalable and robust randomized benchmarking

- of quantum processes,” *Physical Review Letters* **106**, 180504 (2011).
- [39] J. M. Gambetta, A. D. Córcoles, S. T. Merkel, B. R. Johnson, J. A. Smolin, J. M. Chow, C. A. Ryan, C. Rigetti, S. Poletto, T. A. Ohki, M. B. Ketchen, and M. Steffen, “Characterization of Addressability by Simultaneous Randomized Benchmarking,” *Physical Review Letters* **109**, 240504 (2012).
- [40] J. Wallman, C. Granade, R. Harper, and S. T. Flammia, “Estimating the coherence of noise,” *New Journal of Physics* **17**, 113020 (2015).
- [41] A. W. Cross, E. Magesan, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, “Scalable randomized benchmarking of non-Clifford gates,” *npj Quantum Information* **2**, 16012 (2016), [arXiv:1510.02720](https://arxiv.org/abs/1510.02720).
- [42] R. Harper and S. T. Flammia, “Estimating the fidelity of T gates using standard interleaved randomized benchmarking,” [arXiv:1608.02943](https://arxiv.org/abs/1608.02943) [quant-ph] (2016), [arXiv:1608.02943](https://arxiv.org/abs/1608.02943) [quant-ph].
- [43] S. Kimmel, M. P. da Silva, C. A. Ryan, B. R. Johnson, and T. Ohki, “Robust Extraction of Tomographic Information via Randomized Benchmarking,” *Physical Review X* **4**, 011050 (2014).
- [44] C. Ferrie and O. Moussa, “Robust and efficient in situ quantum control,” *Physical Review A* **91**, 052306 (2015).
- [45] D. J. Egger and F. K. Wilhelm, “Adaptive Hybrid Optimal Quantum Control for Imprecisely Characterized Systems,” *Physical Review Letters* **112**, 240503 (2014).
- [46] M. Isard and A. Blake, “CONDENSATION—Conditional Density Propagation for Visual Tracking,” *International Journal of Computer Vision* **29**, 5 (1998).
- [47] C. Granade, J. Combes, and D. G. Cory, “Practical Bayesian tomography supplementary video: State-space state tomography,” <https://goo.gl/mkibt1> (2015).
- [48] S. van der Walt, S. C. Colbert, and G. Varoquaux, “The NumPy Array: A Structure for Efficient Numerical Computation,” *Computing in Science & Engineering* **13**, 22 (2011).
- [49] IPython Development Team, “Ipyparallel,” (2016).
- [50] A. Lee, C. Yau, M. B. Giles, A. Doucet, and C. C. Holmes, “On the Utility of Graphics Cards to Perform Massively Parallel Simulation of Advanced Monte Carlo Methods,” *Journal of Computational and Graphical Statistics* **19**, 769 (2010).
- [51] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, “PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation,” *Parallel Computing* **38**, 157 (2012).
- [52] S. K. Lam, A. Pitrou, and S. Seibert, “Numba: A LLVM-based Python JIT Compiler,” in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM ’15 (ACM, New York, NY, USA, 2015) pp. 7:1–7:6.
- [53] C. E. Granade, *Characterization, Verification and Control for Large Quantum Systems*, Ph.D. thesis (2015).
- [54] J. D. Hunter, “Matplotlib: A 2D graphics environment,” *Computing In Science & Engineering* **9**, 90 (2007).
- [55] T. S. Yu, “Mpltools,” (2015).
- [56] T. M. Cover and J. A. Thomas, *Elements of Information Theory* (Wiley-Interscience, Hoboken, N.J., 2006).
- [57] R. D. Gill and B. Y. Levit, “Applications of the van Trees inequality: A Bayesian Cramér-Rao bound,” *Bernoulli* **1**, 59 (1995), mathematical Reviews number (MathSciNet): MR1354456.
- [58] H. Jeffreys, *The Theory of Probability* (Oxford University Press, Oxford, 1998).
- [59] W. Edwards, H. Lindman, and L. J. Savage, “Bayesian statistical inference for psychological research,” *Psychological Review* **70**, 193 (1963).
- [60] A. M. Johansen, A. Doucet, and M. Davy, “Particle methods for maximum likelihood estimation in latent variable models,” *Statistics and Computing* **18**, 47 (2008).
- [61] C. Ferrie and C. E. Granade, “Likelihood-free methods for quantum parameter estimation,” *Physical Review Letters* **112**, 130402 (2014).
- [62] S. G. Johnson, “PyCall.jl,” (2016).

A Custom Model Example

In Listing 8, below, we provide an example of a custom subclass of `qinfer.FiniteOutcomeModel` that implements the likelihood function

$$\Pr(0|\omega_1, \omega_2; t_1, t_2) = \cos^2(\omega_1 t_1/2) \cos^2(\omega_2 t_2/2) \quad (14)$$

for model parameters $x = (\omega_1, \omega_2)$ and experiment parameters $e = (t_1, t_2)$. A more efficient implementation of this model using NumPy vectorization is presented in more detail in the User’s Guide.

Listing 8: Example of a custom `FiniteOutcomeModel` subclass implementing the multi-cos likelihood (14).

```
from qinfer import FiniteOutcomeModel
import numpy as np

class MultiCosModel(FiniteOutcomeModel):
5
    @property
    def n_modelparams(self):
        return 2

10    @property
    def is_n_outcomes_constant(self):
        return True

    def n_outcomes(self, expparams):
15        return 2

    def are_models_valid(self, modelparams):
        return np.all(np.logical_and(modelparams > 0, modelparams <= 1), axis=1)

20    @property
    def expparams_dtype(self):
        return [('ts', 'float', 2)]

    def likelihood(self, outcomes, modelparams, expparams):
25        super(MultiCosModel, self).likelihood(outcomes, modelparams, expparams)
        pr0 = np.empty((modelparams.shape[0], expparams.shape[0]))

        w1, w2 = modelparams.T
        t1, t2 = expparams['ts'].T

30        for idx_model in range(modelparams.shape[0]):
            for idx_experiment in range(expparams.shape[0]):
                pr0[idx_model, idx_experiment] = (
                    np.cos(w1[idx_model] * t1[idx_experiment] / 2) *
35                    np.cos(w2[idx_model] * t2[idx_experiment] / 2)
                ) ** 2

        return FiniteOutcomeModel.pr0_to_likelihood_array(outcomes, pr0)
```
